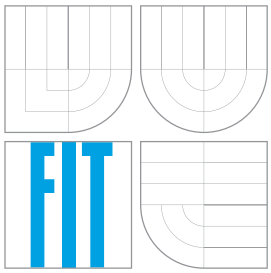
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PORTACE A OPTIMALIZACE ALGORITMU NA DETEKCI ŽIL PRSTU NA PROSTŘEDÍ DSP BLACKFIN

PORTING AND OPTIMIZATION ALGORITHMS FOR DETECTING FINGER VEINS ON DSP BLACKFIN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ADAM TRHOŇ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ MARVAN
SUPERVISOR

BRNO 2011

Abstrakt

Práce se zabývá optimalizacemi obrazového filtru pro extrakci mapy žil prstu ze snímku v NIR spektru a jeho portováním na vývojovou desku s procesorem DSP Blackfin. Při optimalizacích jsou řešeny především mediánové filtry (výsledkem je implementace mediánového filtru s použitím histogramů) a snižování paměťové náročnosti (výsledkem je kvaziparalelní pipeline filtrů). V rámci projektu byla vytvořena i jednoduchá vrstva pro abstrakci hardwaru.

Abstract

This thesis discuss optimizations of algorithms for finger vein map extraction from finger image taken in NIR spectrum. The algorithm is also ported on development kit with DSP Blackfin. For optimization the median filtering was mainly considered (resulting in histogram—based median filter) and the amount of memory used during extraction (resulting in semi—parallel pipeline of image filters). Simple hardware abstraction layer is also implemented.

Klíčová slova

zobrazení žil prstu, optimalizace, obrazový filtr, histogram, medián, paralelizace, DSP Blackfin

Keywords

finger vein image extraction, optimization, image filter, histogram based median, paralelization, DSP Blackfin

Citace

Adam Trhoň: Portace a optimalizace algoritmu na detekci žil prstu na prostředí DSP Blackfin, bakalářská práce, Brno, FIT VUT v Brně, 2011

Portace a optimalizace algoritmu na detekci žil prstu na prostředí DSP Blackfin

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Aleše Marvana. Další informace mi poskytli Ing. Dana Hejtmánková a Ing. Tomáš Novotný. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Trhoň
18. května 2011

Poděkování

Rád bych poděkoval svému vedoucímu, Ing. Aleši Marvanovi, a ostatním členům skupiny STRaDe za jejich vydatnou pomoc a poskytnutí prostředků pro vývoj. Zvláště pak Doc. Ing., Dipl.-Ing. Martinovi Drahanskému, Ph.D za příležitost k vypracování zajímavé a obohacující bakalářské práce.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá optimalizacemi a portováním algoritmu pro extrakci žil prstu z jeho snímku. Výchozím bodem projektu je obrazový filtr VEPA (Vein Extraction Process Algorithm — algoritmus procesu extrakce žil), který ze snímku prstu v NIR (Near Infra Red) spektru extrahuje mapu žil. Filtr je naprogramován v jazyce C jako samostatná aplikace pro použití v terminálu. V rámci práce bude označován jako VEPA SW (software).

Druhým vstupem je deska navržená pro tento projekt, na které je osazen procesor Blackfin ADSP-BF523, 256 Mb SDRAM, IR čidlo pro měření vzdálenosti, 4 LED a řadič pro jejich ovládání, kamera a převodník UART—USB. Deska je označována jako VEPA HW (hardware).

Výstupem projektu by měla být deska pro výukové účely, která pomocí IR čidla detekuje prst nad kamerou, sejme snímek prstu osvíceného LED, extrahuje mapu žil za méně než 0.5 s a výsledek odešle do PC. Výstup projektu je označován jako VEPA FW (firmware).

Práce je rozdělena do pěti kapitol. První kapitolou je Úvod. Druhou je Teoretický rozbor, ve kterém jsou uvedeny vlastnosti detekce žil prstu a přehled použitého hardwaru a nástrojů. Třetí kapitola, Analýza, rozebírá vstupy projektu, úzká hrdla a jiné problémy. V rámci analýzy jsou navrženy optimalizace a výsledná podoba VEPA FW. Kapitola Implementace a testování se zabývá popisem implementace navrženého řešení, jeho testování a profilovými analýzami. V kapitole Závěr jsou zhodnoceny dosažené výsledky.

Kapitola 2

Rozbor

Kapitola obsahuje souhrn teoretických podkladů pro provedení analýzy VEPA SW, implementace a testování VEPA FW.

2.1 Snímání žil prstu

Snímání žil prstu je principem podobné snímání žil dlaně či hřbetu ruky. To je popsáno např. v [12]. Při snímání se požadovaná strana ruky nasvítí IR diodami a její obraz je sejmut kamerou v NIR spektru. Rozmístění žil je v průběhu života stálé a velmi individuální (i mezi levou a pravou rukou či dvojčaty), podobně jako otisk prstu. Oproti jiným metodám má však snímání žil několik výhod. Je bezkontaktní, proto hygienicky čisté. Žíly jsou viditelné díky rozdílům v teplotě oproti okolním tkáním a koncentraci oxyhemoglobinu, je tedy testována i živost. Metoda je navíc odolná vůči lehkým zraněním či onemocněním kůže.

Hlavní výhodou snímání prstu oproti celé ruce je menší hardwarová náročnost sejmutí snímku. Zařízení tak může být kombinováno s jinými metodami, např. snímání otisku prstu, pro zvýšení jejich spolehlivosti.

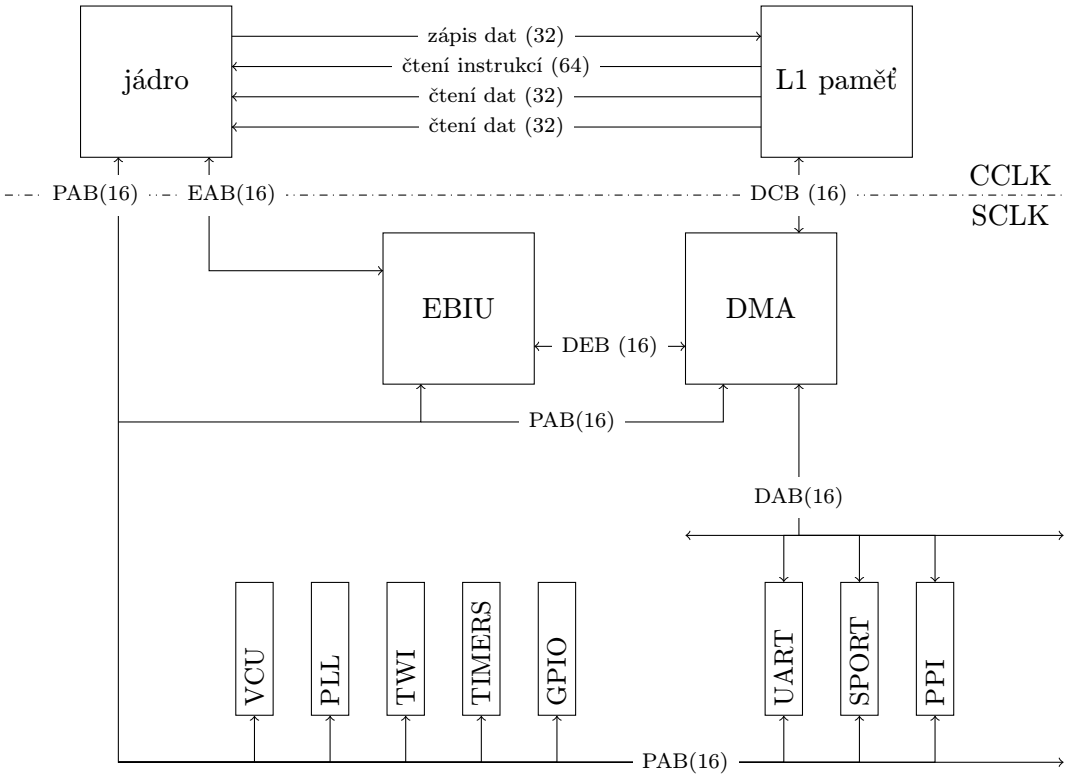
2.2 Architektura procesoru ADSP-BF523

ADSP-BF523 je 32bitový mikroprocesor navržený pro zpracování digitálních signálů. Architektura procesoru je na obrázku 2.1.

Srdcem procesoru je jeho jádro a L1 paměť. Jako jediné části systému pracují na frekvenci jádra (CCLK, až 600 MHz pro ADSP-BF523), zbylé součásti pracují na frekvenci systému (SCLK, až 133 MHz). Propojeny jsou 64 b sběrnicí pro načtení instrukce, dvěma 32bitovými sběrnicemi pro načítání dat a jednou 32 b sběrnicí pro jejich ukládání. Sběrnice mohou pracovat paralelně za předpokladu, že pracují s jinými bloky L1 paměti. Fyzicky se L1 paměť nachází přímo v jádru procesoru a je rozdělena na paměť instrukcí a paměť dat, jedná se o Hardwardskou architekturu.

Ke všem komponentám procesoru přistupuje jádro pomocí sběrnice PAB (Peripheral Access Bus). Spoždění sběrnice trvá 2 cykly SCLK. Synchronizaci (resp. zápis všech operací čekajících na dokončení) umožňuje funkce `ssync()`.

EBIU (External Bus Interface Unit) je zařízení pro komunikaci s externími paměťmi. Přístup jádra do externích pamětí zajišťuje propojení jádra a EBIU sběrnicí EAB (External Access Bus), přístup DMA pak propojení sběrnicí DEB (DMA External Bus).



Obrázek 2.1: Architektura procesoru ADSP-BF523. V závorkách uvedena šířka sběrnic v bitech.

start bloku	velikost bloku	název
0x FFB0 0000	4 kB	Scratchpad
0x FFA1 0000	16 kB	L1 instruction C (cache)
0x FFA0 8000	32 kB	L1 instruction B
0x FFA0 0000	32 kB	L1 instruction A
0x FF90 0000	16 kB	L1 data B (cache)
0x FF90 0000	16 kB	L1 data B
0x FF80 4000	16 kB	L1 data A (cache)
0x FF80 0000	16 kB	L1 data A

Tabulka 2.1: Rozložení L1 paměti v adresovém prostoru

DMA (Direct Memory Access) slouží pro přenos dat mezi periferiemi, interní a externí pamětí bez účasti procesoru. Periferie jsou k DMA připojeny sběrnicí DAB (DMA Access Bus), k EBIU přes DEB (DMA External Bus) a k L1 paměti sběrnicí DCB (DMA Core Bus). Ze schématu je vynechán DMA řadič pro USB (ADSP-BF523 USB modul neobsahuje), který je napojen na sběrnice DEB a DCB.

2.2.1 L1 paměť

Jádro systému má k dispozici celkem 132 kB L1 paměti. Její rozdělení a umístění je uvedeno v tabulce 2.1. Bloky označené jako cache můžou, ale nemusí, být použité pro vyrovnávací paměť. Bloky *instruction* slouží pro uložení programu procesoru, bloky *data* pro uložení dat. V bloku *scratchpad* se nachází zásobník.

2.2.2 Start systému

Při startu systému je spuštěn zavaděč (boot kernel), který se nachází v ROM. Úkolem zavaděče je ze zdroje určeného piny `BMODE[3:0]` načíst firmware formátovaný jako boot stream. Boot stream je rozdělen do bloků, které kromě dat programu obsahují i metainformace, např. cílové umístění nebo příznaky pro řízení načítání.

Jedním z příznaků je přítomnost inicializačního kódu, který se spouští v průběhu zavádění boot streamu a slouží např. pro inicializaci EBIU před načtením firmwaru do SDRAM.

2.3 Vývojový kit ADSP-BF526 EZ-KIT Lite

Vývojový kit ADSP-BF526 EZ-KIT Lite byl použit pro vývoj VEPA FW jako provizorní řešení před dokončením VEPA HW. Použitý procesor (ADSP-BF526) má architekturu podobnou ADSP-BF523 s následujícími rozdíly:

1. Počet jader. ADSP-BF523 je jednojádrový, ADSP-BF526 obsahuje jádra dvě. Pro vývoj VEPA FW druhé jádro nebylo použito.

2. Maximální frekvence jádra je pro BF526 omezena na 400 MHz (ADSP-BF523 může fungovat až na 600 MHz).

3. Rozdílné periferie. Pro projekt VEPA je významnější pouze absence USB modulu na procesoru ADSP-BF523.

Na desce je osazen JTAG emulátor ADZS-DBGAGENT-BRD, který umožňujě nahrávání firmwaru, ladění a profilovou analýzu.

2.4 Sběrnice

Tyto sběrnice a zařízení jsou použity na desce VEPA HW. Uveden je pouze stručný přehled, podrobný popis lze nalézt v jednotlivých standardech a datasheetech.

2.4.1 USB

USB (universal serial bus) je sériová sběrnice pro připojování zařízení k počítači. Každé zařízení je identifikováno číslem výrobce (vendor ID) a číslem výrobku (product ID). Typ zařízení je dán třídou (class), jejíž číslo je přiděleno organizací USB-IF. Architektura je hvězdicová, veškerý provoz je řízen počítačem. Maximální rychlost ve verzi 2.0 je 480 Mbit/s. Podrobnosti o sběrnici lze nalézt v [14].

Sběrnice je tvořena dvěma napájecími vodiči (5 V, 100 mA garantováno, max. odběr po požádání je 500 mA) a dvěma diferenciálními datovými vodiči (D+ a D-). Pro přenos bitů je použito kódování NRZI.

Na sběrnici je jeden master a max. 127 zařízení. Její větvení je zajištěno rozbočovači (hub), jejichž použití je zcela transparentní. Specifikace dovoluje maximálně 7 úrovní větvení.

Veškerá data a informace pro řízení sběrnice jsou přenášena v paketech několika typů (token, data, handshake, start of frame), které jsou kombinovány do jednotlivých přenosů: control (data řízení, kontrola chyb, nemá vlastní pásmo), interrupt (pravidelné doručení malého množství dat, kontrola chyb, vyhrazené pásmo, např. myš), isochronous (časově kritický přenos dat, bez kontroly chyb, vyhrazené pásmo, např. kamera) a bulk (spolehlivý přenos dat, kontrola chyb, nemá pásmo, např. tiskárna).

Aby bylo možné lépe odlišit jednotlivé přenosy, jsou zavedeny roury. Roura (pipe) je abstraktní objekt, který spojuje dva koncové body zařízení (endpoint). Cíl paketu je tak identifikován adresou zařízení a číslem koncového bodu (zde lze nalézt analogii s TCP/IP: adresa zařízení odpovídá IP adrese, číslo koncového bodu číslu portu, roura socketu). Každá roura přenáší pakety jednoho typu (mluví se o typu roury, který je stejný jako typ přenosů) jedním směrem (kontrolní roura je jako jediná obousměrná).

Pro popis zařízení slouží hierarchie deskriptorů. Jedná se o strom struktur, v jehož kořeni je deskriptor zařízení (uvádí např. vendor ID, product ID). Potomky deskriptoru zařízení jsou deskriptory konfigurace (obsahují např. požadovaný proud). O úroveň níže v hierarchii jsou deskriptory rozhraní (obsahují např. třídu, jedno fyzické zařízení tak může implementovat více funkcí, např. tiskárna a scanner). Poslední úrovní jsou deskriptory koncových bodů, které uvádí např. jejich typ, směr či číslo.

Při připojení má zařízení adresu 0. Počítač zařízení detekuje, přidělí novou adresu a vyžádá si hierarchii deskriptorů. Vybere jednu z dostupných konfigurací, zašle zařízení požadavek na její nastavení a začíná zařízení používat. Od této chvíle může zařízení odebírat více proudu než standardních 100 mA (pokud tak uvádí vybraná konfigurace).

2.4.2 SPI

SPI (Serial Peripheral Interface) je sériová synchronní sběrnice určená pro komunikaci mezi integrovanými obvody. V typické podobě je na sběrnici jeden master a několik slave zařízení.

Podporuje libovolné množství zařízení a plně duplexní přenos. Podrobnosti o sběrnici lze nalézt v [1, 8].

Fyzicky je sběrnice tvořena čtyřmi a více vodiči. Dva pro datové signály: MISO (master in, slave out) pro přenos dat ze zařízení master do slave, MOSI (master out, slave in) pro opačný směr. Jeden vodič (SCLK) pro přenos hodinového signálu, jeden a více pro výběr zařízení, se kterým bude master komunikovat (SS, slave select).

Synchronizace hodinovým signálem se nastavuje dvěma bity: CPOL a CPHA. CPOL (clock polarity) určuje polaritu hodinového signálu: pro CPOL=1 je SCLK aktivní v logické 0 a naopak. CPHA (clock phase) určuje, ve které fázi hodinového taktu se data zapisují a ve které čtou: pro CPHA=0 se data zapisují při změně z klidové do aktivní úrovně, čtou při změně z aktivní do klidové a naopak.

Přenos dat začíná nastavením SS cílového zařízení na logickou 0. V každém hodinovém cyklu je přenesen jeden bit dat ze slave do master (signál MISO) a jeden bit opačným směrem (signál MOSI). Po přenosu požadovaného počtu slov (8 nebo 16bitových) je komunikace ukončena nastavením SS na logickou 1.

2.4.3 Inter—Integrated Circuit

I^2C (Inter—Integrated Circuit) je nízkorychlostní synchronní sériová sběrnice vyvinutá firmou Philips. Podporuje propojení až 118 zařízení (pro 7 b adresy), poloduplexní přenos a detekci kolize (dvě zařízení v režumu master na jedné sběrnici) při rychlosti až 100 kbit/s (standardní mód). Podrobnosti o sběrnici (řešení detekce kolize, podmínka RSTART, jiné rychlosti a způsoby adresace) viz. [9].

Fyzicky je sběrnice tvořena dvěma vodiči, SCL pro přenos hodinového signálu, SDA pro datový signál. Oba vodiče jsou napojeny na pull-up rezistor, zařízení jsou připojena technikou otevřeného kolektoru. Zapojení umožňuje detekci kolize (master na SDA zapíše H, High, signál je však ve stavu L, Low) a synchronizaci hodin dle nejpomalejšího zařízení (zařízení drží SCL ve stavu L, ostatní zařízení začínají periodu H až po uvolnění signálu).

Každé zařízení na sběrnici je adresováno 7 b adresou. Adresa 0000000_2 je vyhrazena pro broadcast, adresy 0000011_2, $00001xx_2$ a $11111xx_2$ pro pozdější využití, celkem je tedy možné na sběrnici připojit až 118 zařízení.

Přenos na sběrnici začíná podmínkou START generovanou masterem - během aktivního SCL je SDA přepnut z H na L (spolu s podmínkou STOP jediné povolené případy, kdy se SDA mění při aktivním SCL). Následuje zápis 7bitové adresy a R/W bitu (1 pro read). Pokud některé ze slave zařízení rozpozná svou adresu, po jeden SCL cyklus drží SDA ve stavu L (potvrzení, ACK). Následuje zápis/čtení 8bitových bajtů, každý potvrzený přijímacím zařízením. Pro ukončení přenosu je použita podmínka STOP, kdy je během aktivního SCL přepnut signál SDA z L do H.

2.4.4 UART

UART (universal asynchronous receiver/transmitter) je modul pro sériovou komunikaci. Logické úrovně protokolu odpovídají standardům EIA-232E, EIA-422 či EIA-485, je však potřeba převodník napěťových úrovní. Na desce VEPA je osazen převodník FT2232, který data posílá (přijímá) po sběrnici USB, PC tak komunikuje s deskou přes virtuální sériový port. Rychlost sběrnice je závislá na použitém hardwaru.

Sběrnice je tvořena dvěma asynchronními signály (Rx pro přijímání dat, Tx pro jejich vysílání), které fungují nezávisle na sobě a umožňují tak plně duplexní přenos. Ten začíná

start bitem (logická 0), pokračuje 5 až 8 datovými bity, volitelnou paritou (sudá, lichá, žádná) a jedním až dvěma stop bity.

2.5 Komponenty VEPA HW

Komponenty použité na desce VEPA HW. Jedná se o flash paměť M25P16, LED řadič NCP5680, kameru OV7720, krypto paměť AT88SC3216C, SDRAM MT48LC16M16A2FG-75, převodník úrovní SN74CB3T3125 a USB—UART převodník FT2232HL.

2.5.1 Flash paměť M25P16

M25P16 je sériová flash paměť komunikující přes rozhraní SPI. Podporuje komunikaci na frekvenci až 75 MHz (operace READ je omezena na 33 MHz), SPI módy s CPOL a CPHA rovnou jedné nebo CPOL a CPHA rovnou nule.

Celkem 16 Mb paměti je rozděleno do 32 sektorů po 512 kb nebo 8192 stránek po 256 B. Podporovanými operacemi pro práci s daty jsou PAGE PROGRAM, v rámci které je možné zapsat 1 — 256 B dat do jedné stránky, SECTOR ERASE, která smaže jednu stránku a připraví ji tak pro zápis, BULK ERASE pro výmaz celé paměti, READ pro čtení dat (je možné přečíst celou paměť na maximální frekvenci 33 MHz) a FAST READ pro čtení na frekvenci až 75 MHz. Každý zápis je nutné uvést instrukcí WRITE ENABLE. Dokončení operace zápisu či mazání stránky je možné kontrolovat operací READ STATUS REGISTER. Kompletní přehled a podoba instrukcí je k dispozici v [8].

2.5.2 LED řadič NCP5680

NCP5680 je řadič pro buzení dvojce nebo čtveřice výkonových LED diod. Pomocí násobičky napětí nabije externí kondenzátor, jehož energii následně použije k rozsvícení diod na zadanou dobu. S procesorem komunikuje po sběrnici I^2C jako slave (na sběrnici nikdy nevystupuje jako master). Komunikace probíhá zápisem či čtením sady osmibitových registrů, které jsou uvedené v [10]. V registrech se dá nastavit napětí na LED (stejné pro oba kanály), maximální odebíraný proud na kanálu, šířka pulzu a jeho zpoždění po aktivaci.

Protokol, který řadič používá, je jednoduchý: pro zápis hodnoty do registru je na sběrnici odeslána adresa řadiče pro zápis, osmibitová adresa registru a osmibitová hodnota, která se do registru zapíše. Čtení STATUS registru je speciální případ. Odešle se adresa řadiče pro zápis, adresa STATUS registru a libovolný bajt dat (je ignorován). Po RSTART podmínce se odešle adresa řadiče pro čtení a řadič odpoví osmibitovou hodnotou STATUS registru.

2.5.3 Kryptopaměť AT88SC3216C

Atmel AT88SC3216C poskytuje 4 kB paměti přes sběrnici I^2C. Paměť je rozdělená do 16 zón po 256 B. Pro každou zónu je možné definovat přístupová práva (heslo pro čtení, heslo pro zápis). Přenos dat může být šifrovaný. Podrobný popis lze nalézt v [3].

2.5.4 FT2232H

FT2232H je dvoukanálový převodník ze sériové či paralelní sběrnice na sběrnici USB. Podporuje např. UART (až 12 Mbaud/s, omezeno externím oscilátorem), FIFO, JTAG, I^2C či SPI. Konfigurace je možná pomocí externí EEPROM. Na straně USB zařízení podporuje protokol USB 2.0 High Speed a USB 2.0 Full Speed. Podrobnosti lze nalézt v [4].

2.5.5 MT48LC16M16A2FG-75

SDRAM MT48LC16M16A2FG-75 poskytuje 32 MB DRAM, obvod pro její obsluhu a synchronizaci se systémovým taktem (kompatibilní se standardy PC100 a PC133).

DRAM je dvourozměrné pole paměťových buněk, které svoji informaci ukládají v podobě náboje kondenzátoru. Ty jsou uspořádány do řádků a sloupců, adresa buňky je tak rozdělena na adresu řádku a adresu sloupce. Při zadávání adresy je třeba nejdříve odeslat adresu řádku (je aktivní signál RAS), následně adresu sloupce (aktivní signál CAS). Časový odstup signálů RAS a CAS je označován jako prodleva RAS-CAS (t_{RCD}, RAS to CAS Delay). Po zadání adresy jsou hodnoty buněk v požadovaném řádku přes bitové vodiče přivedeny na zesilovač (viz. dále) a je možné provést čtení či zápis dat.

Protože bitové vodiče mají vyšší kapacitu než paměťové kondenzátory, je třeba speciální postup pro čtení dat. Na bitové vodiče je přivedeno napětí odpovídající polovině maximálního napětí na kondenzátorech. Po ustálení napětí (čas potřebný pro ustálení je označována jako doba přípravného nabíjení RAS, t_{RP}, Row Precharge) je svým adresovým vodičem připojen jeden řádek paměťových buněk. Náboje v kondenzátorech mírně ovlivní napětí na bitových vodičích (hodnota na kondenzátorech je ztracena), rozdíl je zesílen zesilovačem a data je možné číst. Zesilovač zároveň obnoví původní hodnoty paměťových buněk. Čas mezi přivedením adresy sloupce a získáním dat je označován jako přístupová doba CAS (t_{CAS}).

Kvůli vlastnostem kondenzátorů se jejich náboje postupně ztrácí, hodnoty v paměťových buňkách je potřeba neustále obnovovat. K tomu lze využít zesilovač popsaný v postupu čtení. Z tohoto důvodu je také omezena doba Page Mode (v rámci jednoho řádku se pracuje s několika sloupci). Další informace o (S)DRAM pamětech lze nalézt např. v [6], konkrétní hodnoty časování a dalších specifik paměti v [7].

2.5.6 SN74CB3T3125

SN74CB3T3125 je čtyřkanálový převodník napěťových úrovní. Každému kanálu přísluší tři piny: `xA`, `xB` a `xOE` (kde `x` je číslo kanálu). Piny `A` a `B` jsou vstupně-výstupní (je možný obousměrný přenos), pomocí pinu `OE` lze kanál rozpojit. Podrobnosti lze nalézt v [13].

2.6 Architektura knihovny `adi_XXX`

Analog Devices, výrobce procesorů Blackfin, poskytuje IDE (VisualDSP++) s knihovnou pro abstrakci hardwaru. Moduly v knihovně lze rozdělit do dvou skupin: služby systému (system services) a ovladače zařízení (device drivers).

Popis služeb systému lze nalézt v [2]. Primárně se jedná o funkce pro nastavení parametrů systému, jako PLL, VCU, EBIU. Až na výjimky (např. řadič DMA) služby systému nepracují s daty.

Ovladače zařízení naopak primárně slouží pro manipulaci s daty. Rozhraní každého ovladače je tvořeno funkcí pro otevření (volá se pomocí funkce `adi_dev_Open`), konfiguraci (`adi_dev_Control`), čtení dat (`adi_dev_Read`), jejich zápis (`adi_dev_Write`) a uzavření ovladače (`adi_dev_Close`). Programátor s ovladačem nepracuje přímo, o jeho obsluhu se stará správce ovladačů zařízení (device driver manager), jehož dokumentaci lze nalézt taktéž v [2]. Dokumentace k jednotlivým ovladačům se nachází v instalačním adresáři VisualDSP++ (Blackfin/docs/drivers).

Kapitola 3

Analýza

Kapitola se věnuje analýze výchozí verze softwaru (před optimalizacemi a portováním), analýze hardwaru a určení úzkých hrdel.

3.1 Architektura VEPA HW

Na obrázku 3.1 je základní schéma vývojové desky VEPA.

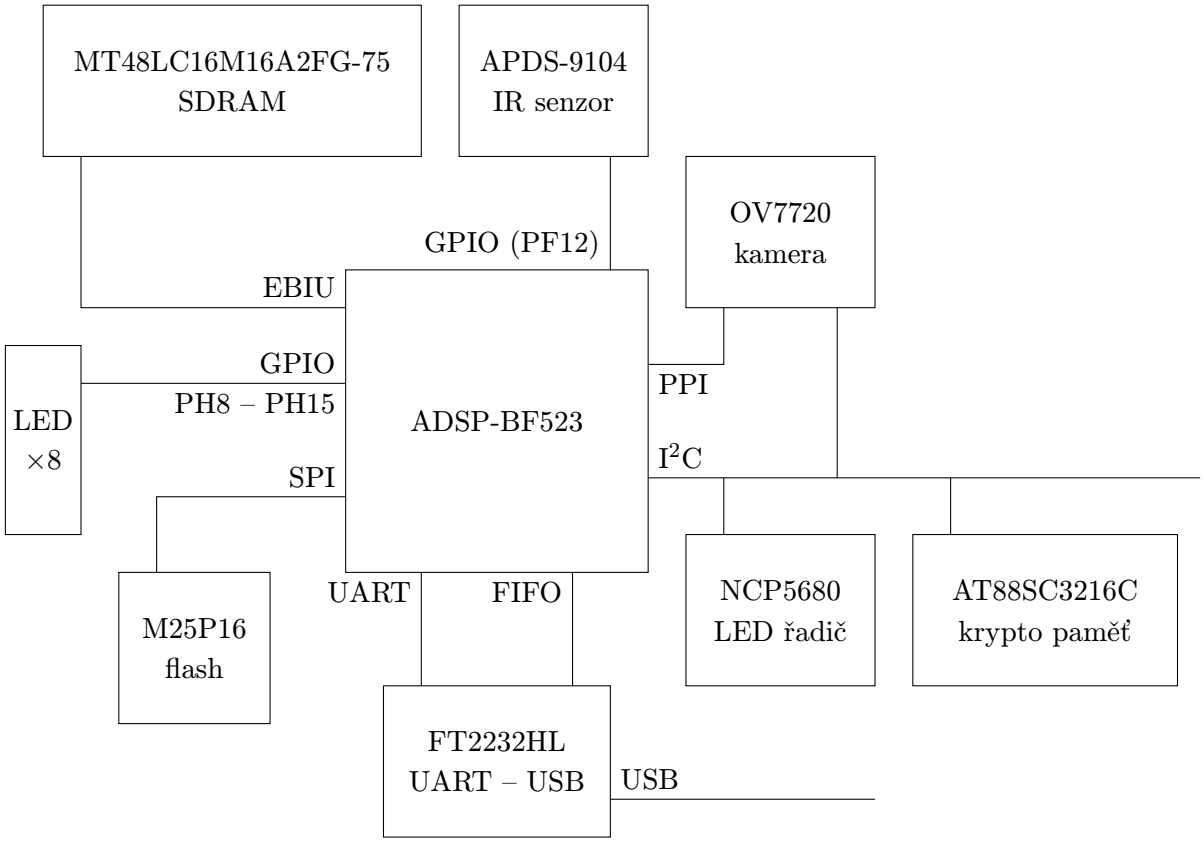
IR senzor je osazen jako detektor přítomnosti prstu nad kamerou. Senzor se skládá z IR LED diody a fototranzistoru. LED je napevno připojena k napájení. Emitor fototranzistoru je připojen na signálovou zem, kolektor spolu s pull-up rezistorem na pin 12 portu F. Díky pull-up rezistoru je klidová úroveň v logické 1, v přítomnosti prstu klesne na logickou 0. Detekce přítomnosti prstu může být provedena detekcí logické 0 nebo sestupné hrany na pinu `PF12`. Kvůli analogové povaze senzoru připojeného na logický vstup mohou vznikat na náběžné i sestupné hraně zákmity. Ovladač senzoru musí tyto zákmity odfiltrovat (kritické zejména na náběžné hraně, tedy sejmutí prstu z přístroje, kdy by se sejmul a zpracoval snímek prázdného prostoru).

Kamera kvůli chybě v návrhu desky (otočený footprint součástky) zůstává neosazena. V rámci další práce na VEPA HW bude zhotovena redukce. Nastavení kamery a synchronizace s osvětlením bude provedena pomocí protokolu SCCB na sběrnici I^2C, přenos obrazových dat bude probíhat na sběrnici PPI.

LED řadič ovládá LED diody osvětlující prst při snímání žil. Nastavení řadiče probíhá přes sběrnici I^2C. Řadič jako takový je určen pro ovládání dvou LED, použitím duálních MOSFET je možné zapojit až 4 diody, které jsou ovládány po dvojcích. Na desce je první dvojce J1 a J2, druhá dvojce J3 a J4. Ovladač musí pro každou dvojci diod nastavit napětí, maximální odebíraný proud, délku a spoždění pulzu po jeho aktivaci. Vzhledem k výkonové povaze subsystému by ovladač měl zajistit alespoň minimální kontrolu nastavovaných dat.

Krypto paměť je posledním zařízením připojeným na sběrnici I^2C. Protokol, který paměť používá, však není kompatibilní s protokolem I^2C. Při čtení dat master odešle požadavek na zápis, 4 bajty adresy, následně paměť začíná odesílat požadovaná data.

Možnosti pro řešení jsou dvě. V rámci první by se softwarově implementoval protokol I^2C pomocí bitů SCLOVR a SDAOVR registru TWI_MASTER_CTL (nastavení hodnot



Obrázek 3.1: Blokové schéma vývojové desky VEPA

na vodičích) a SCLSEN a SDASEN registru TWI_MASTER_STAT (zjištění hodnot na vodičích). Problémem je, že registry jsou součástí master mode funkce a ta konroluje sběrnici kvůli soupeření (bus arbitration). Odpověd paměti by byla detekována jako ztráta sběrnice a činnost master modulu ukončena.

Druhou možností je po zápisu adresy změnit směr toku dat bitem MDIR registru TWI_MASTER_CTL. Toto řešení však není dokumentované, mohlo by vést k nedefinovanému chování.

Flash paměť je použita pro uložení firmwaru. Paměť podporuje varianty SPI protokolu s CPOL = 1 a CPHA = 1 nebo CPOL = 0 a CPHA = 0. Maximální rychlost pro komunikaci je 60MHz (čtení je potřeba provádět operací FAST READ, jinak je max. rychlost 30MHz). Sběrnice SPI, kterou je paměť připojena, je přerušena převodníkem SN74CB3T3125, který pomocí jumperu umožňuje paměť odpojit od procesoru.

Převodník UART — USB slouží pro komunikaci s PC. Připojen přes sběrnice UART (sériová) a FIFO (paralelní), na straně PC ovladač vytvoří dva virtuální sériové porty, pomocí kterých je možné komunikovat s kitem. Jumper P6 umožňuje mezi převodníkem a procesorem vytvořit loopback. Jumpery JP2 a JP3 umožňují přesměrovat USB sběrnici přímo na port procesoru, nicméně ADSP-BF523 modul pro USB nemá.

SDRAM je připojena sběrnicí EBIU. Při jejím nastavení je třeba vzít do úvahy možnosti nastavení frekvencí procesoru. Modul PLL (Phase Locked Loop) umožňuje frekvenci jádra nastavit až na 600 MHz, nicméně frekvence systému je pak omezena na 120MHz. Je nutné použít nastavení pro PC133, paměť je však podtaktována a její výkon snížen. Druhou možností je nastavení hodin jádra na 537.5 MHz a systému na 134.375 MHz. To zrychlí práci s pamětí, na druhou stranu je omezen výkon jádra.

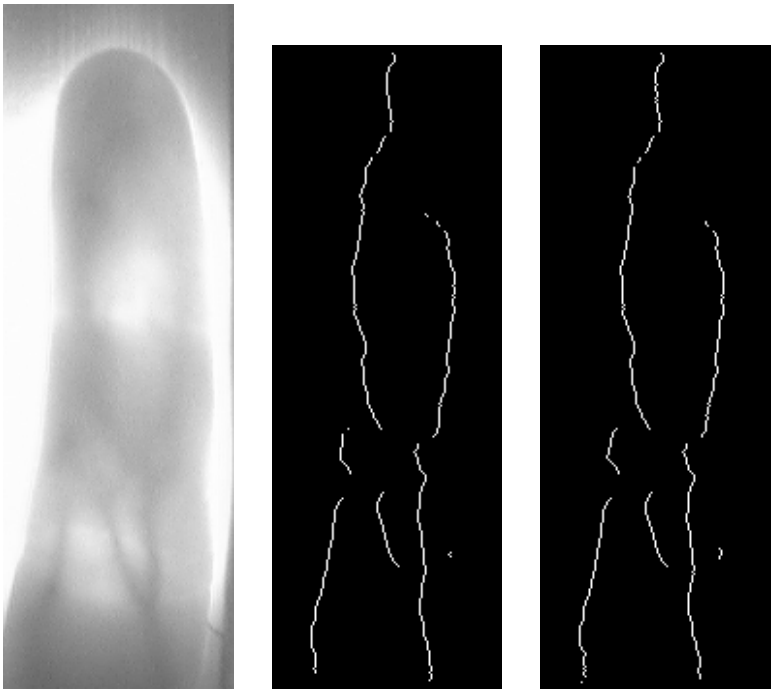
LED jsou zapojeny anodou na napětí 3.3V, katodou na piny 8 – 15 portu H. Rozsvícení diody se provede zápisem 0 na příslušný GPIO pin.

3.2 Architektura VEPA SW

VEPA SW je označení pro software dodaný jako jeden ze vstupů projektu. Jedná se o obrazový filtr, který vstupní bitmapu (24bitový snímek prstu v odstínech šedi) transformuje na mapu žil (24bitů, čenobílá). Příklad vstupu a výstupu je na obrázku 3.2. Obrázek 3.2a je příkladem snímku z IR kamery, 3.2b je výstup originální verze VEPA SW. Obrázek 3.2c je výstup z VEPA FW, který je oproti 3.2b mírně modifikován (viz. dále). Samotný program je složen z několika modulů.

Modul `bmp` načítá a ukládá bitmapy s 24bitovou barevnou hloubkou do souborů formátu Windows Bitmap. Vstup je v odstínech šedi (všechny tři kanály obsahují stejnou informaci), výstup je černobílý (opět všechny kanály stejné). Jednou z optimalizací tedy bude redukce počtu barevných kanálů.

Modul `basic ops` obsahuje implementace základních obrazových filtrů, jako medián, konvoluce, atd. a jejich vývojové varianty. Funkce mají podobnou strukturu. Tři parametry pro velikost, vstupní a výstupní bitmapu. Dvěma zanořenými cykly se prochází jednotlivé pixely výstupní bitmapy a vyhodnocuje se výstup. Pro každý vstupní pixel se kontroluje, zda neleží mimo vstupní bitmapu. Pokud ano, použije se defaultní hodnota (0 coby černý



(a) Originální snímek (b) Výstup VEPA SW (c) Výstup VEPA FW

Obrázek 3.2: Výstupy systému VEPA

pixel, ve všech funkcích stejné). V rámci optimalizací bude počet kontrol mezí polí potřeba zredukovat.

Moduly `finger contour` a `skelet` slouží pro specializovanější operace (extrakce kontury prstu a skeletonizace). Platí stejné závěry jako pro modul `basic ops`.

Při optimalizacích je potřeba zachovat naprosto stejný výstup, což není možné pro operaci konvoluce. Ta pracuje s čísly v plovoucí desetinné čárce. Procesor ADSP-BF523 ale nemá FPU (Floating Point Unit), všechny výpočty by se tak musely náročně emulovat. Proto je třeba výpočet konvoluce převést na pevnou řádovou čárku, čímž se však změní výstup. Spolu s opravou drobné chyby v kontrole mezí bitmapy, kvůli které se nezpracovával jeden řádek a jeden sloupec na okrajích, se jedná o jediné změny na výstupu filtru (viz 3.2c).

3.3 Úzká hrdla

VEPA SW obsahuje několik problematických míst. Tato sekce se věnuje jejich popisu a návrhu různých možností řešení.

Kontroly mezí bitmap

Všechny filtry kontrolují meze polí bitmap. Pokud se jedná o operaci pracující nad maticí, je kontrolován každý prvek matice. Pro nejtypičtější velikost matice (3×3) jedná o 36 kontrol na jeden pixel, náročnost kontrol je tak srovnatelná s náročností samotného filtru. První možností by bylo před procházením matice z její pozice určit, zda může dojít k přístupu mimo bitmapu. Pokud ne, kontroly se neprovádí. Tento přístup však vede k duplicitnímu kódu.

Druhou variantou je alokace ochranného pásma okolo bitmapy vyplněného černou barvou. Kontroly mezí bitmap by tak bylo možné vynechat úplně za cenu mírně větší paměťové náročnosti a nemožnosti definovat různé defaultní hodnoty pro jednotlivé filtry.

Redukce barevné hloubky

Ačkoliv jsou vstupní i výstupní bitmapy 24bitové, všechny tři jejich kanály obsahují stejnou informaci. Protože se každý kanál zpracovává zvlášť, náročnost filtrů je vyšší než je nutné. Pro zvýšení výkonu aplikace a zjednodušení algoritmů bude barevná hloubka snížena z 24 bitů na 8 bitů. Kvůli kompatibilitě s původní verzí a možnosti jednoduchého srovnávání bitmap je vstup a výstup ponechán na 24 bitech.

Duplicitní filtry

Program obsahuje dvě pipeline filtrů, jednu pro zobrazení žil prstu, jednu pro určení jeho kontur. První dva filtry v obou pipeline jsou stejné, proto se budou provádět jen jednou.

3.3.1 Mediánový filtr

Častou z operací je mediánový filtr. Pracuje s maticí, její prvky seřadí dle velikosti a jako výstup vybere střed. Vysoká náročnost filtru spočívá v potřebě vstup seřadit. Optimalizace se dá docílit třemi způsoby. V prvním se mediánový filtr eliminuje tam, kde není potřeba. Ve druhém se spojí s jinou operací a bude optimalizován výsledek. Třetím způsobem je optimalizace samotného filtru.

Eliminace: speciální medián

Funkce pro speciální medián počítá černé pixely ve vstupní matici (3×3). Pokud je černých pixelů (s hodnotou 0) více než 6, výstupem je výstup mediánového filtru. Pokud ne, výstupem je originální pixel. Je zřejmé, že výsledek mediánového filtru, v jehož vstupu je nadpoloviční většina položek nulových, bude 0. Proto je v tomto případě možné výpočet mediánu vynechat a na výstup zapisovat přímo nulu. Tato optimalizace kromě zlepšení výkonu eliminuje výskyt mediánového filtru s maticí o rozměrech 3×3, proto není třeba tento dále uvažovat.

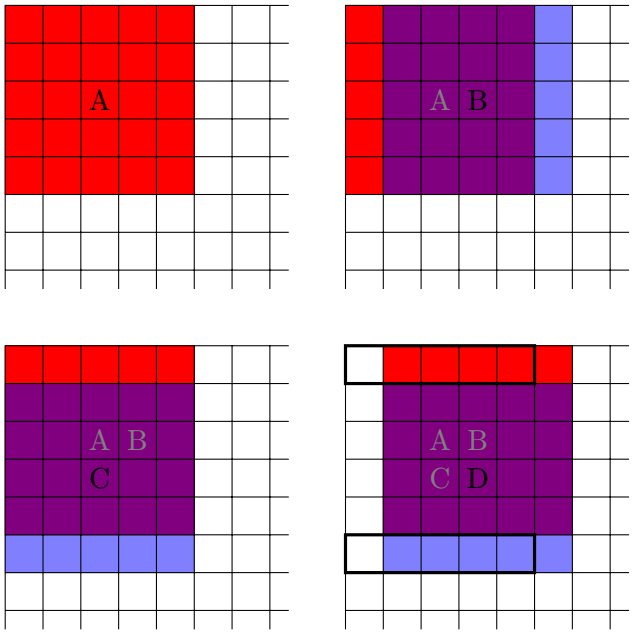
Spojení: prahmedián

Výstupem prahování jsou prvky pouze dvou hodnot. Pro jejich filtrování mediánovým filtrem není třeba vstup řadit, prvky stačí spočítat a jako medián prohlásit hodnotu nadpoloviční většiny. V podstatě se jedná o filtr založený na histogramech, ale protože histogramem je zde pouze jedno číslo, jsou možné další optimalizace, viz. obrázek 3.3.

Černé pixely v matici pixelu A se počítají bez optimalizací.

Matice pixelu B využívá překrytí s maticí A (fialová). Pro výpočet není potřeba procházet celou matici, ale jen ty části matic, teré nejsou překryté - rozdílové vektory (červeně a modře). Stejným způsobem se vypočítá celý první řádek. Počty pixelů se uchovávají v bufferu pro pozdější použití.

Matice pixelu C se počítá podobným způsobem jako matice B, využije se počtu černých pixelů z výpočtu A.



Obrázek 3.3: Princip fungování filtru prahMedian na matici 5×5. Jedno pole mřížky odpovídá jednomu pixelu. A, B, C, D je označení pixelů, pro které se filtr vyhodnocuje. Šedě jsou označeny již vyhodnocené pixely.

Pixel D se počítá stejným způsobem jako pixel C, použijí se rozdílové vektory oproti matici B. K dispozici jsou však rozdílové vektory z pixelu C (na obrázku vyznačené obdélníky), které se s aktuálními vektory z velké části překrývají. Z jejich hodnot je tedy možné minimální úpravou odvodit aktuální rozdílové vektory, výsledkem je 5 operací porovnání (4 pro rozdílové vektory, 1 pro celkovou sumu) a 6 operací sčítání (4 pro vypočítání rozdílových vektorů, 2 pro jejich aplikaci) pro libovolnou velikost matice filtru. Spojením filtrů se navíc eliminuje varianta 7×7, pro další optimalizace zbývá již jen 5×5.

Optimalizace: medián

Jako univerzální varianta se nabízí seřazení prvků pomocí standardní funkce qsort. Ta však prvky porovnává pomocí externí funkce, proto je pro tento případ nevhodná (pomine-li se výhoda snadné implementace).

Druhou alternativou je algoritmus pro výběr k. prvku quickselect [11]. Pracuje podobně jako quicksort, jen po rozdělení vstupu dle odhadu nezpracovává obě části, ale jen tu, ve které se nachází prvek s indexem k. Výsledkem je částečně seřazené pole, ve kterém se k. nejmenší prvek nachází na indexu k.

Další možností (pro filtry 5×5) je algoritmus heap-median [5]. Využívá rychlosti a stability hald pro nalezení 13 největších či nejmenších prvků. Na začátku se inicializují dvě haldy s kapacitou 13 prvků, jedna s minimem, druhá s maximem v kořeni. Do kořene obou hald se uloží první prvek vstupní posloupnosti coby odhad mediánu. Všechny další prvky se zařazují do jedné z hald podle porovnání s odhadem mediánu. Pokud se naplní halda s minimem v kořeni, obsahuje 13 největších prvků z dosud prošlé části vstupní posloupnosti. Zbylé prvky se postupně porovnávají s kořenem této haldy. Pokud je prvek větší než kořen, zařadí se na místo kořene a provede se rekonstrukce haldy. Po zpracování všech prvků vstupu se medián nachází v kořeni haldy (coby nejmenší ze 13 největších prvků). Pro

haldu s maximem v kořeni se postupuje analogicky.

Poslední uvažovanou variantou je mediánový filtr založený na histogramech [15]. Při osmibitové hloubce je histogram tvořen 256 sloupci. Filtr zanese položky do histogramu, jeho postupnou integrací určí medián. Výhodou je konstantní časová náročnost pro zařazení prvku a vyhledání mediánu (vzhledem k počtu prvků). Problém metody spočívá v poměru počtu sloupců (256) ku počtu položek (25). Procházení takto řídkého histogramu je neefektivní. Řešením problému je druhý histogram, do kterého se místo jednotlivých hodnot pixelů zanáší rozsahy po 16 hodnotách. Vzniká histogram se 16 sloupci, z něhož je určen interval, ve kterém se medián nachází. Přesná hodnota mediánu se určí z původního histogramu, ve kterém je potřeba projít maximálně 16 sloupců. Z původního maxima, 256 sloupců, se stává 32 sloupců histogramu, které je potřeba projít. Cenou za optimalizaci je nutnost zařadit hodnotu pixelu do dvou histogramů místo jednoho.

Algoritmus je možné dále optimalizovat využitím překrytí matic, podobně jako prahmedián — viz. pixely A, B na obrázku 3.3. Pokud je nová vyhodnocovaná matice posunuta o jeden sloupec doprava od staré, z histogramu se odeberou prvky v nejlevějším sloupci staré matice, přidají se prvky nejpravějšího sloupce nové matice a histogram je připraven k vyhodnocení.

3.3.2 Snížení počtu přístupů do SDRAM

Architektura původní verze projektu sestávala z jednotlivých filtrů, které byly postupně aplikovány na bitmapy. Tyto filtry k bitmapám přistupují sekvenčně po jednotlivých řádcích, případně k vektoru (v řádu jednotek) řádků.

Na systémové úrovni jsou bitmapy kvůli svým rozměrům (řádově 300 řádků po 100 sloupcích, 30 kB) uchovány v SDRAM. Jelikož přístup do SDRAM je řádově pomalejší než přístup do L1 paměti, vzniká úzké hrdlo a požadavek na jeho odstranění.

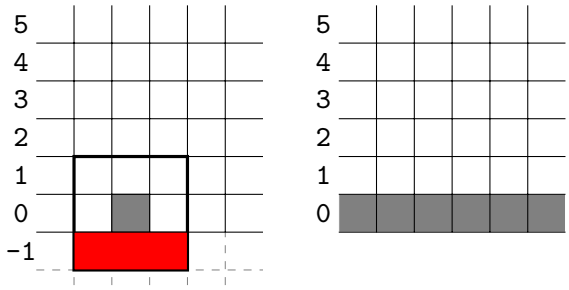
První možností je přesunout obě (vstupní a výstupní) bitmapy do L1 paměti. Kvůli kapacitě L1 paměti, 32kB, tato možnost nepřichází v úvahu.

Druhou variantou je zpracování bitmap po částech. Bitmapa se rozdělí na několik částí, do L1 paměti se načte jedna z nich, je zpracována, uložena a pokračuje se další částí. Nevýhodou je fakt, že při rozdělení bitmapy bude v místech rozdělení docházet k chybám, které postupují hlouběji do bitmapy (viz. obrázek 3.4). Při zpracování návazné části bitmapy je třeba vrátit se o dvojnásobný počet řádků (chyba se vyskytne i v prvních řádcích návazné části).

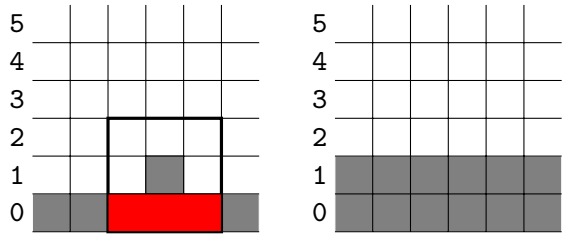
Třetí možností je využití cache. Výhodou této varianty je snadná přenositelnost a jednoduchá implementace. Nevýhodou je fakt, že při aplikaci každého z filtrů je potřeba přenést jednu bitmapu z SDRAM do L1 (vstup) a druhou bitmapu z L1 do SDRAM (výstup), což bude mít za následek velkou režii při přenosu.

Stejné funkcionality se dá dosáhnout čtvrtou variantou, implementací kruhových bufferů ve filtrech. Zpracování bitmap pak probíhá v těchto bufferech. Čtvrtou variantou pak odpadá druhá nevýhoda cache, nutnost synchronizace cache s SDRAM při použití DMA.

Pátá varianta vychází z varianty čtvrté - kruhových bufferů. Pokud operace B následuje po operaci A, je možné výstupní řádek A ukládat přímo do vstupního řádku kruhového bufferu operace B. Výpočet filtru A je přerušen a zpracuje se buffer filtru B. Po dosažení konce pipeline je do SDRAM uložen jeden výstupní řádek, z SDRAM načten jeden vstupní řádek a výpočet se opakuje. Vzniká tak pipeline kvaziparalelních algoritmů, ve které je potřeba jednoho načtení a jednoho uložení bitmapy pro celý výpočet. Pipeline narušuje funkce `linkContour`, která vyžaduje náhodný přístup po celé výšce bitmapy. Výpočet proto



(a) Nejspodnější řádek matice se dostává mimo bitmapu v místě, kde by stále měla být data (záporný index řádku vlevo). Ta jsou nahrazena černými pixely (viz odstranění kontrol mezí polí). Výstup filtru je neplatný (vpravo).



(b) Další filtr v pořadí čte neplatný výstup prvního filtru (vlevo), chyba postupuje hlouběji do bitmapy (vpravo).

Obrázek 3.4: Vznik a postup chyby při rozdělení bitmapy na části

bude rozdělen do třech fází - v první se vyhodnotí pipeline, ve druhé funkce `linkContour`, ve třetí se dle kontury ořízne výstup.

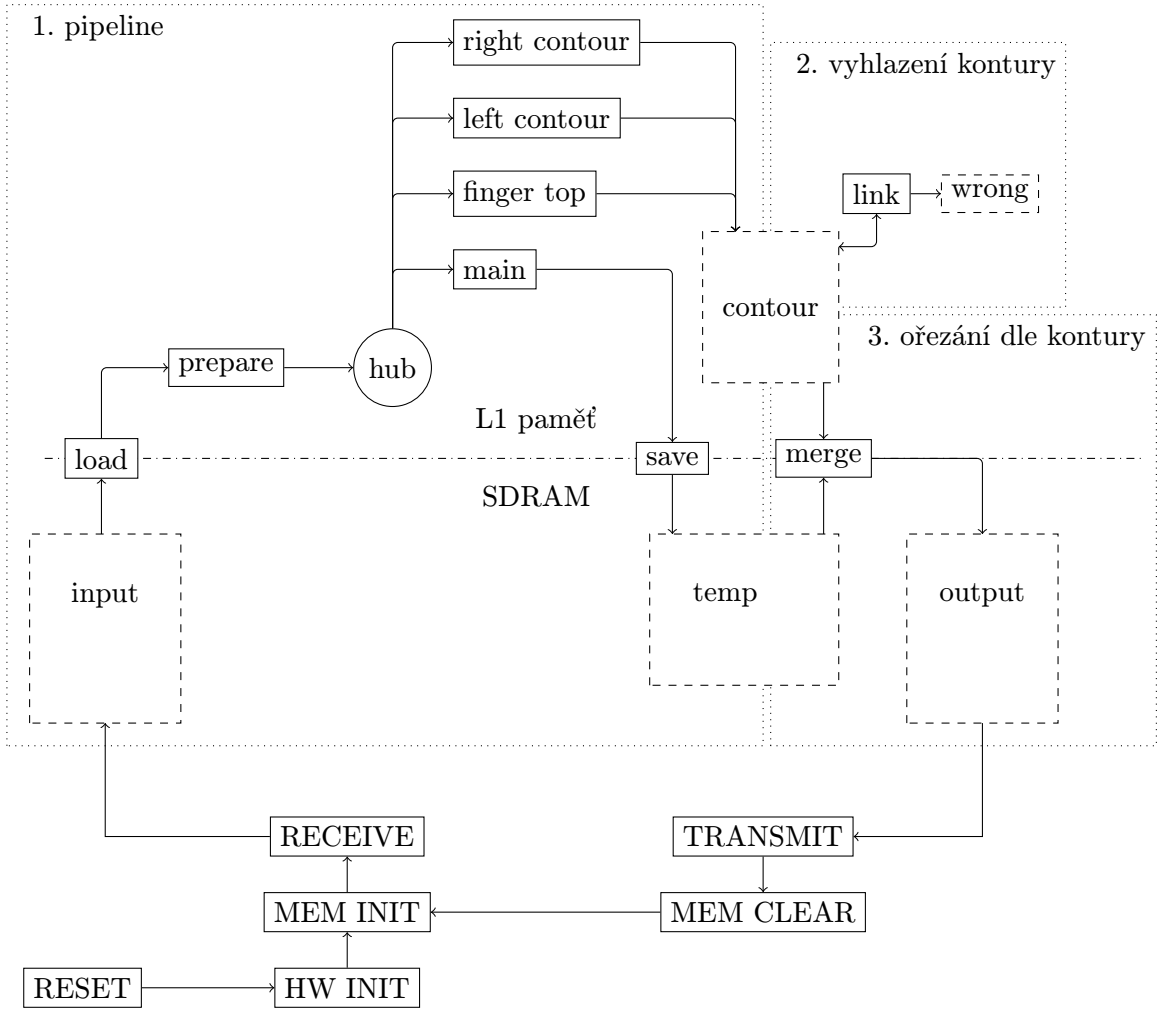
Kromě pipeline byla zvážena i varianta rozdělení bitmap. Ta by teoreticky mohla dosahovat stejných či mírně lepších výsledků s jednodušší architekturou, její nevýhodou je však vysoká paměťová náročnost. Pro co největší efektivitu je třeba bitmapu rozdělit na co nejmenší počet částí a maximálně tím naplnit L1 paměť, což bude problém, pokud se projekt v budoucnosti použije v komplexnějším zařízení (např. s operačním systémem), ve kterém bude aktivována cache.

3.4 Architektura VEPA FW

Na obrázku 3.5 je návrh výsledné podoby VEPA FW. Po resetu systému je provedena inicializace hardwaru. Následný kód již probíhá v nekonečné smyčce. Je inicializován alokátor bitmap, přijata bitmapa, zpracována (tři fáze zpracování: pipeline, vyhlazení kontury, ořez výsledku) a odeslána jako odpověď do počítače. Smyčka končí úklidem alokátoru bitmap, který je v následujícím běhu opět inicializován. Pro profilovou analýzu jsou významné pouze tři fáze zpracování, proto se nepočítá inicializace a úklid alokátoru.

3.5 HAL

V rámci portování VEPA SW na VEPA HW je potřeba vytvořit jednoduchou vrstvu pro abstrakci hardwaru. Knihovna poskytovaná s VisualDSP++ není zcela vyhovující kvůli



Obrázek 3.5: Návrh výsledné podoby VEPA FW. Plnou čarou procesy, čárkovaně datová úložiště. Tečkovaně jsou vyznačeny jednotlivé fáze, čerchovaně oddělena SDRAM od L1 paměti.

své robustnosti (pro projekt tohoto rozsahu příliš vysoká) a velikosti výsledného firmwaru (situace je mnohem jednodušší, pokud se firmware vejde do L1 paměti — maximálně 80 kB). Proto byl pro implementaci výsledné podoby VEPA FW vybrán opensource sada překladačů gcc pro procesory Blackfin.

Pro dosažení maximální jednoduchosti architektury a minimální velikosti výsledného firmwaru není při návrhu kladen maximální důraz na obecnost a formální čistotu. Příkladem je modul `UART`, ve kterém je třeba nastavit děličku hodin systému tak, aby výsledkem byl požadovaný baudrate. Formálně správným postupem by bylo volání funkce z modulu `power`, která vrátí aktuální frekvenci. Ta se použije pro odvození hodnoty děličky. To by však vedlo k poměrně velkému množství kódu, který je potřeba pouze při inicializaci. Proto bude dělička nastavena ručně spolu s instrukcemi pro změnu uvedenými v komentáři.

3.5.1 UART

Modul `UART` bude sloužit pro komunikaci VEPA HW s počítačem: v současné verzi projektu (bez kamery) příjem snímku prstu a odeslání odpovědi — mapy žil. Protože během komunikace není potřeba provádět jinou část programu, nejsou využita přerušení ani DMA. Modul by měl obsahovat funkce pro čtení a zápis jednoho znaku, přečtení zadaného množství znaků, zápis zadaného množství znaků a zápis řetězce.

3.5.2 SDRAM

V modulu `SDRAM` je potřeba jediná funkce — inicializace. Vzhledem k minimálnímu množství přístupů do SDRAM je z dvou možností v kapitole 3.1 možné jednoznačně vybrat variantu s vyšším výkonem jádra.

3.5.3 POWER

V modulu pro správu napájení a hodinových taktů je opět potřeba pouze inicializace. Jak je uvedeno v kapitolách 3.5.2 a 3.1, vybrána je varianta s vyšším výkonem jádra: CCLK = 600 MHz, SCLK = 120 MHz. V budoucnu by modul mohl být rozšířen o dynamickou správu, uspávání jádra při komunikaci s PC či čekáním na vstup, atd.

3.5.4 SPI

Protože na sběrnici SPI se nachází jediná komponenta, flash paměť, budou funkce spojeny do jediného modulu — `spi_flash`.

Při každé transakci na sběrnici SPI se jeden bajt dat zapíše a jeden přečte, modul SPI procesoru ADSP-BF523 proto obsahuje dva datové registry: vstupní a výstupní. Průběh transakce je ovlivněn jejím startem. První možností je start při čtení vstupního registru. Obslužná funkce musí zapsat data k odeslání, přečíst data ve vstupním registru a čekat na dokončení (návrh počítá se synchronními operacemi). Druhou možností je start při zápisu do výstupního registru. Oblsužná funkce zapíše data k odeslání, počká na průběh transakce a vrátí obsah vstupního registru.

Z hlediska kompaktnosti a přehlednosti kódu se lépe jeví druhá varianta. První by dovolila poloasynchronní řešení (zadá se transakce, funkce se ihned vrátí, na dokončení transakce se čeká při dalším zadání), což by za určitých podmínek mělo pozitivní dopad na výkon, nicméně vzhledem k povaze návrhu (bez paralelizace) a účelu modulu (flash loader) to není potřeba.

3.5.5 Flash paměť

Jak bylo zmíněno v kapitole 3.5.4, modul pro flash paměť je spojen s modulem pro SPI. Každý z příkazů uvedených v datasheetu [3] bude implementován jako samostatná synchronní funkce.

3.5.6 I²C

Modul pro I²C bude sloužit ke komunikaci s LED řadičem a kamerou. Modul musí podporovat základní transakce na I²C: odeslání start/rstart/stop podmínky, příjem bajtu (s potvrzením i bez) a odeslání bajtu. Pro podporu kamery musí modul obsahovat i funkce komunikující protokolem SCCB.

3.5.7 LED řadič

LED řadič je ovládán zápisy do svých registrů, je proto vhodná obecná funkce pro zápis hodnoty do registru. Pro čtení STATUS registru je třeba implementovat vlastní funkci. Pro alespoň částečné zajištění bezpečnosti budou všechny parametry buzení zadány jedním voláním funkce (nemůže dojít k opomenutí některého z nastavení) a jejich hodnoty kontrolovány. V přípaně nebezpečného zadání (např. příliš vysoký výkon) pak nebude provedeno rozsvícení diod.

Kapitola 4

Implementace a testování

4.1 Použité nástroje

Práce na PC

Na PC byl pro překlad použit překladač gcc verze 4.5.2, pro profilovou analýzu gprof 2.21. Automatické řízení analýzy zajišťoval skript napsaný pro bash. Skript testoval správnost výstupu optimalizovaného filtru pro každou bitmapu programem diff, (porovnání výstupu původní a optimalizované verze) a vypisoval výsledky profilové analýzy.

Vývojový kit EZ-KIT Lite

Pro vývoj na EZ-KIT Lite s procesorem BF526 byla použita zkušební verze programu VisualDSP++. Komunikace s PC probíhala po USB sběrnici, bylo potřeba vyvinout klientský program (napsán v C, pro práci s bitmapami byly k dispozici knihovny z VEPA SW), modul pro Linux (na základě ukázkového modulu skeleton, který je dostupný ve zdrojových kódech Linuxu) a jednoduché pravidlo pro udev pro nastavení přístupových práv k zařízení.

Na straně kitu byly použity knihovny dostupné v rámci VisualDSP++, pro ovládání USB modulu ovladač implementující třídu `bulkadi`. Ta obsahuje dva koncové body (vstupní a výstupní) typu bulk. Ovladač pro Linux pak umožnoval komunikaci podobnou sériovému portu.

Správnost výstupu se testovala pro každou bitmapu programem diff (automatizace skriptem pro bash). Statistickou profilovou analýzu zajištovalo prostředí VisualDSP++. Po dodání desky VEPA HW byly práce na EZ-KIT Lite ukončeny.

4.1.1 VEPA HW

Pro vývoj na VEPA HW byl použit gcc toolchain pro procesory Blackfin. Komunikace s PC probíhá po sběrnici USB. Na desku byl osazen převodník FT2232C, softwarově je tedy možné komunikovat pomocí rozhraní UART. Komunikaci s kitem zajišťuje skript `vepa-client.py` napsaný v Pythonu, použita byla knihovna pySerial a PIL. Skriptu se předávají tři parametry: soubor zařízení, vstupní bitmapa a výstupní bitmapa. Profilovou analýzu zajišťuje firmware sám. Testování výstupu srovnáváním s výstupem referenční verze provádí skript `vepa-test.py`. Pro srovnání bitmap slouží skript `diffimg.py` (jako parametry se předávají jména souborů dvou bitmap), který extrahuje obrazová data a srovnává pouze ta (narozdíl od programu diff, který srovnává celé soubory). Neuvažují se tak rozdíly v hlavičkách či formátu bitmapy.

Bootování

VEPA HW umožňuje dva typy bootování: boot z UART a boot z flash paměti. Který z módů je použit určují jumpery `JP[789]`. Ty jsou připojené na piny `BMODE[013]` tak, že při propojení všech jumperů systém bootuje z flash, při rozpojení z UART.

Protože ADSP-BF523 nepodporuje zápis bootovacího streamu do flash paměti, byl vytvořen flash loader. Ten se nahraje přes UART pomocí programu bfin-elf-loader, spustí se, z PC přijme požadovaný firmware a zapíše ho do flash paměti. Na straně PC je pro komunikaci s loaderem k dispozici samotný bfin-elf-loader. Flash loader pak musí simulovat funkci boot kernelu a odesílat odpověď autobaudu (při správně nastavených rychlostech v kitu a PC se autobaud provádět nemusí). Jako mezipaměť může být použita SDRAM — flash loader nahraje boot stream z rozhraní UART a ukládá ho do SDRAM. Po přijetí celého streamu se určí velikost firmwaru, smaže potřebný počet stránek flash paměti a provede zápis.

Profilová analýza

Protože JTAG emulátor ICE-100 neumožňuje profilovou analýzu, firmware ji provádí sám. Primární účel analýzy je srovnání několika verzí jednoho filtru, stačí tak analyzovat běh jednoho filtru a celé pipeline. Funkce analýzy na začátku a konci měřených úseků vyčítá hodnotu časovače jádra. Při zasílání výsledků funkce odvodí z frekvence jádra a uložených počtů period čas běhu měřeného filtru a celé pipeline.

Výběr profilované funkce

Modul profilové analýzy obsahuje čtyřznakový globální řetězec, který označuje profilovanou funkci. Řetězec je makrem preprocesoru převeden na globální 32bitové ID. Funkcím analýzy je předávána lokální čtyřznaková řetězcová konstanta identifikující úsek, se kterým funkce pracuje. Řetězec je převeden na lokální 32bitové ID úseku. Pokud se globální a lokální ID shodují, funkce započne či ukončí měření úseku. Použití konstant a inline funkcí zajišťuje minimální overhead na začátku a konci profilovaného úseku, nulový mimo něj.

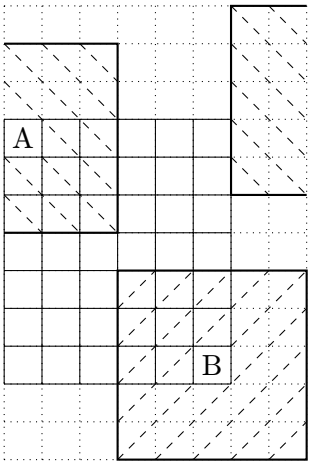
Varianta funkce se vybere konstantou preprocesoru ve zdrojovém souboru dané funkce — pro medián je to soubor median.c, pro prahmedián soubor prahmedian.c.

4.2 Optimalizace režie

Do optimalizace režie patří ty optimalizace, které přímo nesouvisí s prováděnými výpočty. Mezi ně patří převedení interní reprezentace bitmap a s ní pracujících algoritmů z 24 b hloubky na 8 b (popsané v 3.3), odstranění dynamické alokace a kontrol mezí polí, snížení počtu přístupů do SDRAM.

4.2.1 Odstranění dynamické alokace

Systémová knihovna ADSP-BF523 obsahuje funkce dynamické alokace, nicméně pro zjednodušení (odpadá nutnost kontrolovat výsledky alokace) a lepší možnost automatizace ochranného pásma (viz. 3.3) byl sestaven modul implementující vlastní režii dynamické alokace (modul memory). Funkce `init_bitmap_memory()` pro inicializaci interních struktur, `alloc_bitmap()` pro alokaci bitmapy, `free_bitmap()` pro její uvolnění. Pro kontrolu uvolnění všech alokovaných bitmap slouží funkce `terminate_bitmap_memory()`.



Obrázek 4.1: Implementace ochranného pásma pro matici 5×5

Vlastní implementace pak sestává z globálního datového úložiště a zásobníku ukazatelů na volné bitmapy (obojí viditelné pouze z modulu `memory`). Pomocí konstant v `memory.h` lze definovat maximální rozměry bitmap, maximální počet bitmap a bitovou hloubku.

4.2.2 Odstranění kontrol mezí bitmap

Implementaci ochranného pásma navrženého v kapitole 3.3 znázorňuje obrázek 4.1. Pro přetečení se definují řádky a sloupce navíc (matice B). Podtečení (matice A) využívá toho, že v jazyce C se řádky polí ukládají v paměti za sebou, přistupuje se tedy k pixelům na konci předchozího řádku bitmapy. Pásmo zajišťuje funkce `alloc_bitmap()`, maximální velikost aplikovatelné matice se zadává konstantou v souboru `memory.h`.

4.2.3 Snížení počtu přístupů do SDRAM

V rámci projektu je implementována pátá varianta řešení problému — pipeline. Funkce, kterým se předávaly dvojce bitmap, jsou přepsány na funkce, které pracují s kruhovými buffery a při jednom zavolání zpracují jeden řádek bitmapy. Aby bylo možné filtr použít na více místech, veškeré informace, které jsou potřebné mezi zpracováním jednotlivých řádků (kruhový buffer, počitadlo zpracovaných řádků…) jsou uloženy v tzv. kontextu. Kontext je struktura, která je spolu s ukazatelem na výstup (řádek v kruhovém bufferu dalšího filtru) předána obslužné funkci filtru. Dohromady tato trojice (funkce, kontext, výstup) tvoří proces.

Problémy, které bylo potřeba při návrhu řešit, byly dva: spoždění filtrů a větvení pipeline. Větvení pipeline je implementováno pomocí procesu hub (rozbočovač). Poslední proces větvené bitmapy svůj výstup zapisuje na vstup rozbočovače a pipeline je ukončena. Rozbočovač jako takový je tvořen jednořádkovým bufferem, který je čten na začátku větví obslužnou funkcí rozbočovače. Protože obslužná funkce rozbočovače je volána víckrát pro jeden řádek, není v ní možné automaticky inkrementovat počitadlo řádků.

Druhým problémem je spoždění filtrů. Každý proces, který pracuje s víceřádkovým bufferem, musí nejdříve naplnit buffer vstupními daty, platný výstup je generován až poté. Kontrola platnosti výstupu je zajištěna při obsluze pipeline: porovná se spoždění filtru a počet přijatých řádků bitmapy (obojí uloženo v kontextu procesu) a pokud není generován

platný výstup, pipeline je restartována. Z tohoto důvodu je potřeba i možnost povolení a zakázání pipeline jako takové - větev se povolí až po zapsání prvního platného řádku do rozbočovače.

Zpracování dat po vyčerpání vstupní bitmapy je řešeno jak v obsluze pipeline, tak v obsluze procesu. Obsluha procesu musí zajistit, aby po zpracování celé bitmapy (počet zpracovaných řádků je porovnán s výškou bitmapy) na výstup generovala pouze řádky vyplněné černou barvou. To je důležité i pro následující filtr - nekontrolují se meze bitmapy, při přístupu mimo ni musí být čteny pouze černé pixely (viz. ochrané pásmo v kapitole 3.3). Obsluha pipeline se nesmí zakázat dříve, než její poslední proces zpracuje všechny řádky bitmapy.

Při návrhu struktur kontextů byly do jisté míry využity principy dědičnosti. Bázová struktura by obsahovala počitadlo řádků, spoždění a ukazatel na vstup (jeden z řádků bufferu). Obsluha pipeline přistupuje pouze k těmto položkám, se zbylými položkami se pracuje jen ve funkcích pro inicializaci a obsluhu procesu. Volání procesu a kontrolu platnosti dat tak bylo možné sdružit do makra preprocesoru, které významně zpřehlednilo zdrojový kód.

4.3 Optimalizace mediánových filtrů

Coby nejnáročnějšímu filtru [úvodní profilová analýza] byly na mediánových filtrech provedeny nejvýraznější optimalizace. Prvním krokem byla eliminace filtrů tam, kde pro výpočet nejsou potřeba (speciální medián). Druhým krokem spojení s jinou operací (prahmedián) a optimalizace celku, třetím pak optimalizace zbylých filtrů.

4.3.1 Eliminace: Speciální medián

Filtr byl optimalizován dle návrhu — byl kompletně vypuštěn výpočet mediánu. Při testování prošel testem správnosti výstupu.

4.3.2 Spojení: Prahmedián

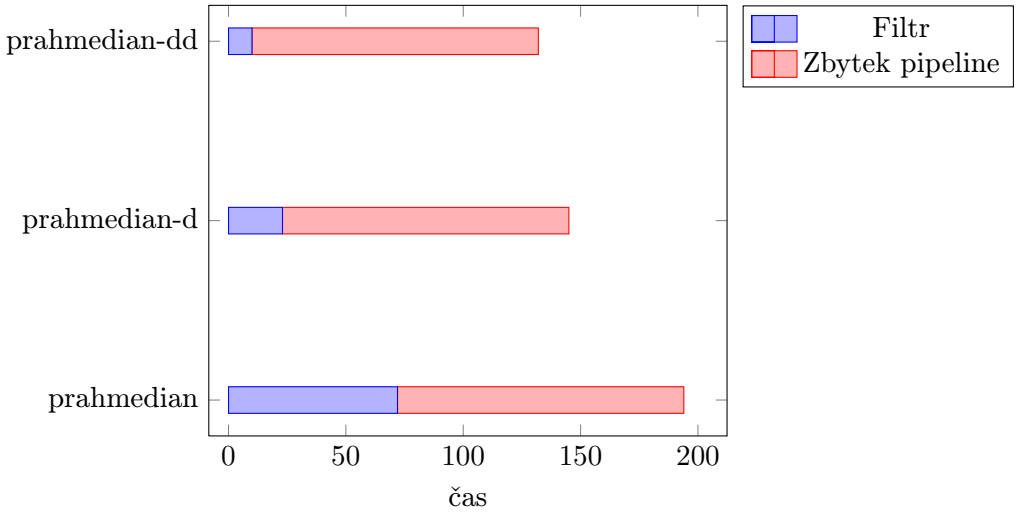
Filtr byl optimalizován tak, jak je popsáno v návrhu. Profilová analýza (tabulka 4.1 a obrázek 4.2) dokazuje zisk rychlosti pro jednotlivé optimalizace: verze s překrytím rozdílových vektorů (prahmedian-dd), verze využívající překrytí matic (prahmedian-d) a verze bez překrytí (prahmedian). Optimalizace prošla testem správnosti výstupu pipeline.

4.3.3 Optimalizace: `median`

Pro optimalizaci zbyl jediný mediánový filtr s maticí 5×5. U tohoto počtu prvků kromě vlastní náročnosti algoritmu rozhoduje i režie, proto byly pro seřazení testovány i jednoduché algoritmy, např. select-sort. Dalšími testovanými variantamy byl bubble sort (coby původní řadicí metoda použitá v prototypu programu), quicksort (pomocí standardní funkce `qsort`), quickselect[11], heap-median[5] a histogram-median.

heap-median

Metoda při testech na PC fungovala rychleji než quickselect (dle testů autora jen pro odpovídající počet prvků - pro 29 má již být quickselect rychlejší), nicméně v testech na ADSP-



Obrázek 4.2: Srovnání variant prahmediánu

varianta	$pipeline[ms]$	$filtr[ms], \sigma$	$\frac{filtr}{pipeline}$	$\frac{filtr}{reference}$
prahmedian	194.9	72.0, 0.0	0.369	1
prahmedian-d	145.3	23.0, 0.0	0.158	0.319
prahmedian-dd	132.0	10.0, 0.0	0.076	0.014

Tabulka 4.1: Srovnání variant filtru prahmedian. Varianta bez optimalizací je považována za variantu referenční.

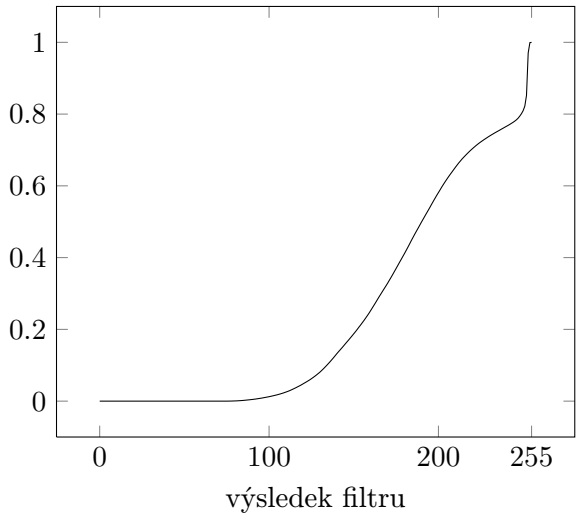
BF523 se časová náročnost ukázala být vyšší. Výhodou metody je její stabilita (worst-case přibližně odpovídá best-case).

histogram-median

Pro výpočet mediánu byla použita metoda histogramů. Kromě optimalizací v návrhu byl navíc změněn směr procházení histogramů. Skript `histogram.py` aplikuje mediánový filtr 5×5 pro všechny vstupní bitmapy a vykreslí histogram výsledků. Ten ukazuje, že výsledky mediánového filtru se pohybují v horní polovině rozsahu - hodnoty 100 - 255. Proto je výhodnější procházet histogram od hodnoty 255 k hodnotě 0 (viz. distribuční funkce rozložení na obrázku 4.3). Vzniká varianta `histogram median reverse`. Na základě rozložení výsledků histogramů bude možné provést doladění algoritmu pro kameru osazenou na VEPA HW. Kromě změny směru procházení je možné i zvýšit počet histogramů (rychlejší konvergence k výsledku vs. nutnost vícekrát zařazovat prvek) nebo změnit šířky sloupců (za předpokladu, že součin šířek bude roven 256).

Testování

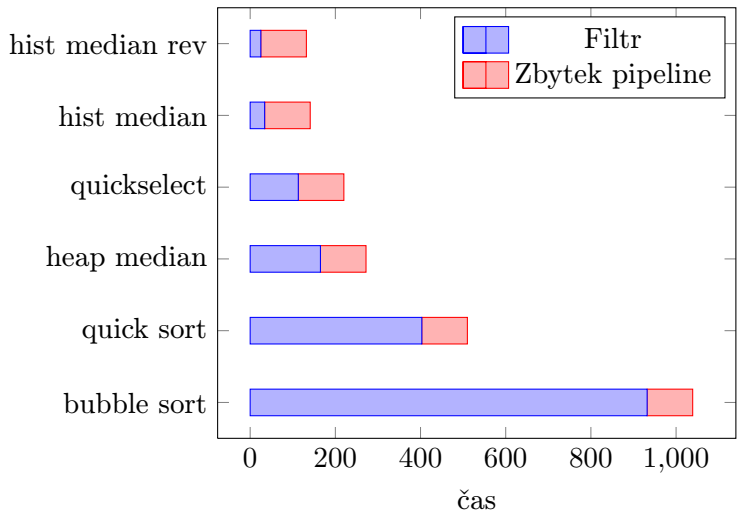
Tabulka 4.2 a graf 4.4 srovnávají výsledky profilové analýzy jednotlivých variant filtru. Všechny varianty prošly testem správnosti výstupu pipeline.



Obrázek 4.3: Distribuční funkce rozložení pravděpodobnosti výsledků mediánového filtru 5×5 aplikovaného na testovací snímky

varianta	$pipeline[ms]$	$filtr[ms], \sigma$	$\frac{filtr}{pipeline}$	$\frac{filtr}{reference}$
bubble sort	1039.5	932.9, 1.197	0.897	8.24
quick sort	510.4	403.7, 5.258	0.791	3.56
heap median	272.2	165.5, 1.167	0.608	1.46
quickselect	220.1	113.2, 0.533	0.515	1
hist median	141.0	34.3, 0.558	0.243	0.30
hist median rev	132.0	25.3, 0.577	0.192	0.22

Tabulka 4.2: Porovnání variant meidánového filtru. Referenční variantou je quickselect.



Obrázek 4.4: Srovnání variant mediánového filtru

4.4 Výsledná podoba VEPA FW

VEPA FW byl implementován dle návrhu. Profilová analýza neukázala významné ztráty na výkonu ve funkci merge, která pracuje s daty přímo v SDRAM. Protože v době provádění třetí fáze již není potřeba vstupní bitmapa, bitmapy input a output jsou uchovány na jedné adrese.

Datové úložiště pro alokátor bitmap je umístěno v SDRAM. Díky použitému toolchainu a minimální velikosti HAL se zde jedná o jediný objekt (vše ostatní se vejde do L1 paměti). Jako takový je pomocí ukazatele umístěn na adrese 0x 0000 0000 (počátek adresového rozsahu SDRAM), což umožnilo výrazné zjednodušení procesu překladu a startu systému. Při standardním umístění prostředky jazyka C by byla kromě úpravy skriptu pro linker potřeba inicializační funkce pro modul EBIU, která se spouští v průběhu příjmu firmwaru. Během provádění funkce musí být příjem pozastaven, což kvůli nezapojenému signálu RTS-CTS na UART sběrnici znemožnuje start systému přes toto rozhranní. S úložištěm umístěným pomocí ukazatele je zcela postačující, pokud není úložiště použito před inicializací EBIU modulu.

VEPA FW komunikuje pomocí sběrnice UART s baudrate 921600. Nejdříve je přijata velikost ve formátu `w%3dh%3d`. Na jejím základě je alokována vstupní bitmapa. Jsou přijata data bitmapy v binární podobě a proběhne výpočet. Po dokončení výpočtu jsou odeslána dvě pětimístná čásla - čas provádění pipeline v milisekundách a čas provádění proflilované funkce. Po časech je odeslána velikost výstupní bitmapy (nemusí být stejná kvůli ořezání ve funkce `merge`) a binární data bitmapy. Zkontroluje se uvolnění všech bitmap a čeká na přenos nového vstupu.

Kapitola 5

Závěr

V práci jsou pokryty všechny body zadání. Byly nastudovány podklady (architektury vývojových desek, datasheety komponent, algoritmus detekce žil, viz. bod zadání č. 1), do provozu uvedeny potřebné vývojové nástroje (VisualDSP++ a GNU Toolchain, viz. bod 2), proveden rozbor algoritmů a jejich úzkých hrdel (s přihlédnutím na použitý hardware). Po základních optimalizacích byl software portován na vývojovou desku (bod 3) a dále optimalizován (bod 4). Testování přínosu optimalizací a správnosti výstupů (bod 5) bylo prováděno průběžně. Výsledky jsou popsány také průběžně, shrnuty a zhodnoceny v rámci závěru.

Výsledkem práce na projektu VEPA je vývojový kit, který snímek prstu zpracuje za cca 120 ms a výsledek odesílá do počítače. Výstup algoritmu je téměř shodný s referenčním výstupem. Změny jsou způsobeny rozdílností hardwaru a vedoucím byly označeny jako nepodstatné. Kvůli chybě v návrhu desky nebylo možné použít kameru, snímek je tedy nutné poslat kitu z PC. Kromě kryptopaměti, která porušuje protokol sběrnice I^2C, a neosazené kamery byly zprovozněny všechny součásti desky. Nahrávání firmwaru je s GNU nástroji možné přes sérovou linku přímo do paměti procesoru (nikoliv SDRAM), nebo s použitím flash—loaderu (vyvinut v rámci projektu) do flash paměti. Kit s počítačem komunikuje pomocí skriptu napsaného v jazyce Python a pro usnadnění dalšího vývoje umožňuje jednoduchou profilovou analýzu. V rámci pokračování práce na projektu bude zhotovena redukce pro připojení kamery a vytvořen firmware podle zadání. Po dokončení bude mít projekt význam jako pomůcka pro výuku biometrie či jako základ pro podobné projekty.

Literatura

[1] Analog Devices: *ADSP-BF52x Blackfin Processor Hardware Reference*. 2010, http://www.analog.com/static/imported-files/processor_manuals/BF52xHRM_Rev.1.0.pdf.

[2] Analog Devices: *Device Drivers and System Services Manual for Blackfin Processors*. 2010, http://www.analog.com/static/imported-files/software_manuals/50_dd_ss_proc_man_rev4.0.pdf.

[3] Atmel: *CryptoMemory Specification*. 2007, www.atmel.com/dyn/resources/prod_documents/doc8664.pdf.

[4] FTDI: *FT2232H DUAL HIGH SPEED USB TO MULTIPURPOSE UART/FIFO IC*. 2009, www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232H.pdf.

[5] Meessen, C.: Median value selection algorithm [online]. http://www.disnetwork.info/1/post/2009/06/median-value-selection-algorithm.html, 2009.

[6] Messmer, H.-P.; Dembowski, K.: *Velká kniha Hardware*. CP Books, 2005, ISBN 8025104168, 1224 s.

[7] Micron: *256Mb: x4, x8, x16 SDRAM*. http://download.micron.com/pdf/datasheets/dram/sdram/256MSDRAM.pdf.

[8] Numonyx: *Numonyx Forté Serial Flash Memory M25P16*. 2010, http://www.micron.com/get-document/?documentId=5976.

[9] NXP: *UM10204, I^2C Bus specification and user manual*. 2007, http://www.nxp.com/documents/user_manual/UM10204.pdf.

[10] ON Semiconductor: *NCP5680*. 2009, www.onsemi.com/pub/Collateral/NCP5680-D.PDF.

[11] Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; aj.: *Numerical recipes in C*. Cambridge: Cambridge University Press, druhé vydání, 1992, xxvi+994 s., the art of scientific computing.

[12] Rak, R.; Matyáš, V.; Říha, Z.: *Biometrie a identita člověka*. Grada, 2008, 664 s.

[13] Texas Instruments: *SN74CB3T3125 QUADRUPLE FET BUS SWITCH 2.5-V/3.3-V LOW-VOLTAGE BUS SWITCH WITH 5-V-TOLERANT LEVEL SHIFTER*. 2006, http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=sn74cb3t3125&fileType=pdf.

[14] USB-IF: *USB 2.0 Specification*. 2000, http://www.usb.org/developers/docs/usb_20_021411.zip.

[15] Weiss, B.: Fast Median and Bilateral Filtering [online]. http://www.shellandslate.com/fastmedian.html.

Příloha A

Obsah přiloženého CD

Na CD přiloženém k práci se nachází:

- Zdrojové kódy výchozí verze programu (adresář `vepa-sw/`) včetně záplat
- Zdrojové kódy aktuální verze firmwaru (`vepa-fw/`)
- Zdrojové kódy programu pro zápis firmwaru do flash paměti (`flash-loader/`)
- Spustitelné verze programů vyvinutých v rámci projektu (`bin/`)
- Použitý GNU toolchain (`blackfin-toolchain-elf.tbz`)
- Upravená verze programu `bfin-elf-ldr` (`bfin-elf-ldr-patched/`)
- Sada skriptů pro práci s projektem (`scripts/`)
- Instalační skript (`install.sh`)
- Programová dokumentace (`doc/`)
- Soubor `README` s podrobným popisem jednotlivých částí

A.1 Záplaty VEPA SW

Ve výchozí podobě není VEPA SW přeložitelný na GNU/Linuxu, je přiložena malá záplata pro řešení problémů (`vepa-sw-linux.patch`). Druhá záplata (`vepa-sw-conv.patch`) upravuje funkci konvoluce na celočíselnou aritmetiku. Úpravy jsou rozděleny pro možnost srovnání původního a nového výstupu. Aplikují se programem `patch` automaticky při instalaci.

A.2 Úprava `bfin-elf-ldr`

Při použití programu `bfin-elf-ldr` spolu se skripty v Pythonu docházelo k neznámé chybě na systémové úrovni. Ta vedla k odhalení chyby ošetření čtení ze sériového portu právě v `bfin-elf-ldr`. Protože chyba doposud nebyla oficiálně opravena, CD obsahuje verzi s možnou opravou (viz. `bfin-elf-ldr-patch/README`).

A.3 Sada skriptů

V adresáři `scripts/` se nachází skripty pro práci se systémem VEPA. Ty se spouští z kořenového adresáře projektu (např. `./scripts/vepa-test.py`). Skripty slouží ke komunikaci s kitem, automatizaci testování a zpracování výsledků profilových analýz. Podrobnosti v souboru `scripts/README` a jednotlivých skriptech.

A.4 Instalační skript

Skript `install.sh` slouží k vytvoření lokální kopie vývojového prostředí pro GNU/Linux. Spouští se z kořenového adresáře CD. Jako parametr je mu předán cílový adresář, ve kterém je vytvořen adresář s prostředím (`trhon-vepa`). V průběhu instalace skript zažádá o kopii testovacích snímků do nově vytvořeného adresáře pro vytvoření referenčních výstupů. Pokud snímky dodány nebudou, výstupy se nevytváří a skript pokračuje dál. Podrobnosti viz. `README` a samotný skript.